ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing

Yuvraj Agarwal
University of California, San Diego
yuvraj@cs.ucsd.edu

Malcolm Hall
University of California, San Diego
mhall@cs.ucsd.edu

ABSTRACT

In this paper we present the design and implementation of ProtectMyPrivacy (PMP), a system for iOS devices to detect access to private data and protect users by substituting anonymized data in its place if users decide. We developed a novel crowdsourced recommendation engine driven by users who contribute their protection decisions, which provides app specific privacy recommendations. PMP has been in use for over nine months by 90,621 real users, and we present a detailed evaluation based on the data we collected for 225,685 unique apps. We show that access to the device identifier (48.4% of apps), location (13.2% of apps), address book (6.2% of apps) and music library (1.6% of apps) is indeed widespread in iOS. We show that based on the protection decisions contributed by our users we can recommend protection settings for over 97.1% of the 10,000 most popular apps. We show the effectiveness of our recommendation engine with users accepting 67.1% of all recommendations provide to them, thereby helping them make informed privacy choices. Finally, we show that as few as 1% of our users, classified as experts, make enough decisions to drive our crowdsourced privacy recommendation engine.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; D.4.6 [**Operating Systems**]: Security and Protection—*Privacy*;

General Terms

Design, Measurement, Experimentation, Human Factors

Keywords

Privacy, Mobile Apps, Crowdsourcing, Recommendations, Apple iOS

MobiSys'13, June 25-28, 2013, Taipei, Taiwan

1. INTRODUCTION

Smartphones are rapidly becoming the mobile platform of choice for users worldwide. While there are several factors behind this growth, one key driver is the "App Store" model of software distribution introduced by Apple, Google, Microsoft and others. Using standardized APIs third-party developers can build applications for platforms such as iOS and Android, which has enabled even those with limited programming experience to release Apps. The phenomenal impact of this model is clear – Apple surpassed 40 billion App downloads in Jan 2013 with over 775,000 iOS Apps in their App Store.

However, the popularity of this model is also a cause for concern, especially from a privacy perspective. In order to build apps with rich functionality, some of the APIs provided by smartphone platforms provide access to potentially private user data. For example, access to the users location, address book, music, photos and unique identifiers (UDID, IMEI,Wi-Fi MAC etc) can be used to track users across applications. Managing access to this private data has become more important given the change in the development model. Until a few years ago, mobile apps were written by a few trusted companies. In contrast, they are now implemented by many individual developers, small companies and large entities alike - not all of whom can be trusted. Furthermore, a large fraction of apps are free, where developers try and supplement their income by incorporating third party advertisement frameworks such as Apple's iAd or Google's Admob. Rather than the app's code, often it is the ad frameworks that access private user data to show targeted ads [14, 27].

The two popular smartphone platforms, Android and iOS, have different approaches towards tackling this problem and for reviewing apps in general. Android does not specifically review apps for the data they access, and instead requires developers to specify the permissions their application needs, notifying the user about them at install time. As a result users bear the responsibility of looking at the permissions and either accepting *all*, or opting not to use the app. In contrast, Apple does review each app submission to make sure that it meets the App Store Review guidelines and any violations lead to rejection. Until recently, iOS did not notify users of apps accessing private information. iOS 5 added notification and control over apps accessing location, while iOS 6 (releasing in September 2012) adds it for contacts, calendars and photos.

Despite these provisions, it has been shown that access to privacy sensitive information by 3rd party apps is rampant

in both Android [3, 7, 8, 14, 26, 31] and even in iOS[6]. While in some cases these accesses are indeed warranted for the functionality of the app, in others there may not be an apparent reason. A recent incident causing public outcry was when Path, a social networking app on iOS, was found to access and upload users address books unknown to them. In a few cases, apps access private data and transmit it for downright malicious reasons [19, 29, 28]. Recognizing these privacy breaches, researchers have utilized static analysis [6, 30], or run time checks on suitably modified versions of the Android OS [7, 14, 31] to detect access to private data. Recently, researchers have proposed substituting shadow data in place of private data [3, 31], or blocking the transmission of sensitive information over the network [14].

Unfortunately, almost all of the prior research focuses on the Android platform due to its open source nature as well as the number of restrictions on iOS (Section 2 provides background on iOS). The sole exception is PiOS[6] which utilizes static analysis to show that numerous iOS apps do access private user information. Furthermore, its still not clear how users react to prompts or cues about privacy and whether the protections provided by smartphone OSes are effective in enabling users to make informed choices. In a recent study, Felt et al. [11] observed and interviewed users to evaluate their understanding of Android permissions noting that very few users actually looked at, or even understood, the permissions requested by applications while others just accepted them by default.

In this paper we present **ProtectMyPrivacy** (PMP), a system that detects access to private data at runtime by apps on iOS devices. Similar to prior work on Android, PMP enables users to decide fine grained privacy protection settings on a per app basis by allowing suitably anonymized data to be sent instead of privacy sensitive information [3, 14, 31]. However, a crucial differentiator of PMP as compared to prior work is that we have developed a crowdsourced system to collect and analyze users protection decisions to help make privacy recommendations to users. The key issue addressed by our crowdsourcing feature is to determine if a clear consensus emerges among the users of an app whether it *should* be granted access to private user data for its functionality, or for a user visible feature or a clear need. We show that our crowd sourced recommendations help our users make more informed privacy choices. Our hope is that with the fine grained privacy control enabled with PMP the balance of power on what private information apps have access to will shift from being in the hands of developers, to the users. As a result, developers will have increased incentives to educate and inform the user about the information they collect and why the users should agree to provide it.

In this paper we make the following contributions:

- The design and implementation of ProtectMyPrivacy, a system for iOS devices to detect access to private information and protect users by substituting anonymized data based on user decisions. A central component of ProtectMyPrivacy is a crowdsourced recommendation engine to analyze manual protection decisions, and use them to provide app privacy recommendations.
- We deployed PMP to real users via the Cydia Store and have collected data on their protection decisions over the past nine months. We present a large scale characterization of access to private data by over 225,685 iOS apps used by 90,621 users of PMP (as of Dec 10th, 2012). Our data shows that 48.4% of apps access the identifier, 13.2% user location, 6.2% user contacts and 1.6% the user's music library.
- We further evaluate PMP by analyzing over 5.97 million protection decisions made by our users, with an average user making 66 decisions. Based on these decisions, we can provide privacy recommendations for 97.1% of the most popular apps (top 10,000). Finally, we show that our recommendations are effective with over 67.1% of all recommendations presented to our users being accepted by them.

2. IOS SECURITY MODEL AND 'JAILBREAKING'

Before describing ProtectMyPrivacy, we briefly discuss Apple iOS, focusing on its security model and privacy features. iOS uses code-signing, encryption and sandboxing to secure the platform. Code-signing ensures only executables reviewed and distributed by Apple are allowed to run. Encryption ensures apps can't be reverse-engineered, protecting the app developer's investment, and that only the purchaser can launch the app. Sandboxing ensures that individual apps cannot access others' data or other parts of the file system. Apple provides various APIs to communicate with the OS and allow apps to register a URL handler that allows them to communicate with each other using parameters. Apps can access shared resources, including privacy sensitive data such as contacts, location, photos using well defined iOS APIs. Recognizing the need for protecting user privacy, Apple introduced pop up notifications in iOS 5 for when an app requests the user's location allowing users to also deny access. In iOS 6, this privacy notifications has been expanded to include contacts, calendars and photos however it is yet to be seen if these many notifications will become obtrusive.

Apps are normally distributed via the Apple controlled App Store and they are subject to a review process to make sure they adhere to strict guidelines. The review includes static analysis to ensure only the published API methods are used and runtime analysis to check apps don't attempt to read outside of the sandbox. However, due to the sheer number of apps submitted, apps that secretly circumvent guidelines sometimes pass review and go on sale; however these are often later removed.

A point of some discussion is the level of control Apple places on the App Store and on the iOS platform. For an app to be distributed in the App Store, developers must use only the published APIs in the Apple SDK, or risk the app being rejected. In response to this an alternative distribution channel has become popular - known as the Cydia Store - which also supports the distribution of apps that function outside of Apple's guidelines, for example those that use lower level capabilities than the public SDK offers in addition to OS 'tweaks' and customizations. The Cydia community actually began before there was even an official App Store, and since has become quite popular, with 4.5 million active users in April 2011[15] which increased to 14 Million active users on iOS6 alone in March 2013 [24] with an estimated 23 Million total jailbroken iOS devices. The use of Cydia requires modification of an iOS device using a process called *jailbreaking* which removes Apple's code-

signing requirement, such that apps can be installed from other sources, *in addition to* installing standard apps from the regular App Store.

While jailbreaking has been deemed legal in the US, it is not supported by Apple since it is often used to install pirated apps and more importantly circumvent iOS protection features. It is therefore possible that jailbreaking may leave users vulnerable to other attacks if they install untrusted apps from unknown sources rather than just from the App Store. As we will describe later in this paper, PMP requires the ability to intercept calls to official iOS APIs that apps use to access private information such as user location or contacts. Unfortunately, there is no way to do this on non-jailbroken devices due to the platform security. Therefore, in order for us to do this large scale app privacy study we had to develop PMP on jailbroken devices and leverage the Cydia store to distribute PMP.

The overall implications and the security discussion of jailbreaking are beyond the scope of our paper. We also do not advocate jailbreaking as a method to protect user privacy, although a number of our users did contact us to say that they jailbroke their devices just to use PMP. We also note that for the purpose of our study, we considered apps that users install from the App Store and not any of the Cydia apps. Although it's possible that some of the apps we have tracked have been pirated, these are essentially decrypted, but identical, versions of purchased apps, and so far no apps have been seen that have been tampered with further. As a result, we believe our findings generalize to the apps that normal non-jailbroken users would install from the App Store.

3. RELATED WORK

Related work on smartphone privacy falls into four general categories: mechanisms of data access control, studies of privacy issues, mechanisms to mitigate privacy issues, and finally user perceptions about app privacy.

Most research into data access control has been explicitly developed for the Android OS in which apps explicitly ask for permissions from the user at install time. Avik [5], ScanDroid [13] and Kirin [9] look at expressing the security properties of Android Apps formally [5], reason about whether extracted permissions are upheld [13], or analyze them for cases when combined permissions can be dangerous [9]. Barrera et al. [1] analyze over a thousand Android apps showing that only a small fraction of requested permissions are used widely. Stowaway [10] similarly analyzes over a thousand Apps statically to show that a third request one or more extra permissions than they really need. Note, the above papers explore retrofitting the Android permission model to determine extraneous permissions but do not explicitly protect user privacy from malicious applications.

A first step towards protecting user privacy is detecting apps that access sensitive data. TaintDroid [7] proposes modifying the Android OS such that 'taint' values can be assigned to sensitive data and their flow can be continuously tracked through each app execution, raising alerts when they flow to the network interface. TaintDroid imposes a runtime overhead because it runs continuously for all applications and hence the authors tested it on a set of thirty popular Android apps reporting that many of them leak privacy sensitive data. In contrast, PiOS [6] employs static analysis techniques to detect privacy leaks. The authors of PiOS construct a CFG of downloaded and decrypted iOS apps, to determine if there is a path from 'sources' of sensitive data to 'sinks' where the information can leave the device. Their analysis on 1400 iOS apps detected over a hundreds apps accessing and sending the UDID, and less often the location and the address book. The Spyphone project [22] reported a number of private data elements accessible by apps, however we chose to study the ones that seemed most privacy intrusive as the basis for our work. The Wall Street Journal also did a study in 2010 on fifty popular iPhone and Android applications each and analyzed their network traffic to detect privacy leaks [25, 26] (as has also been done by others [23]). While these approaches detect privacy breaches by apps, they do not provide mechanisms to actually protect the user from them.

To protect user privacy, it is important to understand when apps access private user data for legitimate reasons such as to provide location based services, and distinguish it with questionable reasons such as sending data to ad networks [6, 27] or downright malicious reasons [19, 29, 28]. Researchers have therefore proposed methods to prevent privacy sensitive data from being acquired by apps in the first place. The Apex system [20] extends Android to enable users to selectively allow, deny or constrain access to the specific permissions requested by applications. A side effect of denying access to resources is that an app may throw an exception and terminate. Mockdroid [3] proposes modifying the Android OS so as to substitute private data with *mock* data – such as constant values or null values – when apps request it. The authors ran MockDroid on the same set of 27 apps used by the TaintDroid [7] system and showed that most of them continued to function, but with reduced functionality in some cases. Zhou et al. [31] similarly developed TISSA for Android that provides various types of mock data in place of private data at runtime to untrusted apps based on user preferences. The authors evaluated TISSA on 24 free Android apps showing that it imposes minimal overhead on a modified Android OS [31]. Appfence [14] builds upon the TaintDroid system [7] to provide shadow data to untrusted apps as well as perform exfiltration blocking to prevent sensitive data from leaving the device. Appfence focuses on the user visible side-effects of giving shadow data, and their evaluation on fifty Android apps showed that two thirds of the effective permissions could be reduced without affecting the app functionality. A limitation of the prior approaches is that they need a modified version of the Android OS. To address this criticism Jeon et al.[16] developed two tools, "Dr. Android" and "Mr. Hide", that run on stock Android phones. Mr. Hide is a service that provides access to private data, while Dr. Android is a Dalvik Rewriter tool that retrofits existing Android apps to use the interface exported by Mr. Hide. Their analysis of this on a set of 19 apps demonstrated that these techniques worked well, although for certain applications the rewriting did change the behavior of the app itself. Finally, the PrivacyBlocker application [30] uses static analysis on application binaries and replaces calls for private data with hard coded shadow data. This approach however requires target applications to be re-written and reinstalled and cannot be done at runtime.

Researchers have begun to explore user perceptions of privacy on smartphones. Felt et al. [11] interviewed Android users on their understanding of Android permissions noting that very few users actually looked at them and often accept

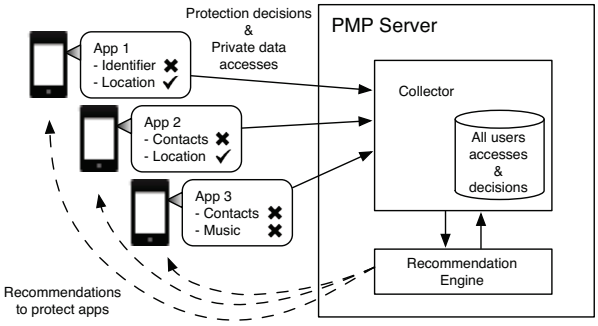


Figure 1: ProtectMyPrivacy architecture based on crowd sourcing. Participating devices transmit their protection decisions for individual applications to our server, which processes the data and uses that to provide recommendations to other devices and users.

them by default. In a project contemporary to ours, Lin et al. [17] collect users expectations of the permissions that particular apps may require, and then measure the users responses to the actual private data access by the same apps using Taintdroid. The authors recruited 179 Amazon Mechanical Turk users to crowdsource these decisions for the top 100 Android apps, observing that users were often surprised about the accesses to their private data and unable to determine why it was needed. In a similar vein to PMP, the paper proposes a new privacy summary screen to give users additional privacy context about apps during installation. While PMP also employs crowdsourcing for app privacy decisions, we focus on building and deploying a scalable recommendation system and evaluating its efficacy on a large number of real users.

PMP differs from the prior work in several key areas. First, as far as we know, except PiOS [6] all of the prior work has been for Android. Android, or the Dalvik VM it uses, being open source is amenable to modification while iOS is more complicated because of its closed source nature and native compiled code. Second, most prior work has been tested on a somewhat limited set of apps, and more importantly not on real users in the wild. In contrast, PMP has been deployed to 90,621 users with over 225,685 unique apps. This paper is therefore the first large scale study showing the actual extent of access to private data by iOS apps as well as our users' response to it by protecting and allowing these accesses. Finally, all of the prior works, which propose replacing privacy sensitive data by shadow data, rely on users to make the appropriate decisions. While PMP also provide users a similar ability to make these decisions on their own, our recommendation engine feature that employs crowd sourcing to suggest protection settings to users is novel and has not been used before, or evaluated at scale, in this context.

4. ARCHITECTURAL DESIGN

We had a number of goals in mind when designing ProtectMyPrivacy (PMP). First, PMP should be architected such that it can be evaluated by a large population of real smartphone users, despite its technical complexity. Second, PMP should be able to detect runtime access to private information, and determine whether to allow or deny the access (either by prompting the user or automatically). Third, PMP should have an easily accessible user interface (UI) for configuring privacy settings across apps. Finally, we wanted to design PMP such that we can collect protection decisions made by users and feed this data into a recommendation engine so that future users benefit from crowdsourced protection decisions.

Most of the prior work on smartphone privacy for Android proposed either using modified versions of Android running on a few test devices or using static analysis on a set of downloaded apps. The number of apps that can be tested are limited in both cases since new apps and updated versions are released often. In contrast, we designed PMP to be fully dynamic in nature, running in the hands of real iOS users "in the wild". Standard iOS apps from the App Store are self-contained and sandboxed, that is they can only access data and settings created by that particular app. However, PMP should run transparently as a plug-in to any unmodified app that the user runs, whilst communicating with its own database to store settings.

PMP detects access to private data and allows the user to protect or allow these accesses, as illustrated in Figure 1. The first information type we protect is the smartphone's unique device identifier (UDID), which is a SHA1 hash of a concatenated string of unique hardware id's including the serial number, International Mobile Equipment Identity (IMEI), Wi-Fi MAC and Bluetooth MAC. Developers can utilize the UDID to monitor app usage, advertising networks can utilize it for cross-app tracking, associating the identifiers with user location and behavior [27]. The number is also linked by the carrier to personally identifiable billing information such as name and address.

The next private data type that PMP protects is the user's address book stored in a database on the device, containing contact information such as names, addresses, phone numbers and emails. Recently, the Path social networking app was found to read the users' entire address book and upload it to a server without their permission, causing significant backlash. It is not certain if Path, or other apps that do this, profit from the sale of personal information, but it is possible that this information (such as address books) could be sold by unscrupulous developers. PMP should empower users to allow access to their address book, or protect it by redirecting the access to an alternative address book, filled with fictitious entries (names, emails and phone numbers) on a per app basis. Sending fictitious information not only protects the user but also reduces the integrity of the rogue developers' remote database, making it difficult to distinguish between genuine and fictitious data, reducing the value and potentially preventing a sale.

A long standing concern is location privacy. If an app does not require location for an obvious or needed feature then PMP should allow the user to protect their location. For the same reason, as with the UDID, PMP should provide a random location or allow users to choose a fake location to prevent profiling. We note that Apple has indeed recognized the need for protecting user privacy and added notifications for access to contacts, calendar and photos in iOS 6 (released Oct 2012). However, since the list of private data items that apps can access may increase, PMP should be extensible in the data types it protects, so that our system can simply be updated to accommodate them. These include apps accessing photos, music libraries, and other ways of tracking

user identity such as the MAC addresses, device name, user email address and phone number.

An important aspect of detecting and manipulating runtime access to private information is that PMP needs to have a very low overhead such that it does not affect app or device performance or reduce battery life. Furthermore, since apps may access private data at any point during execution, PMP should pause the app's runtime when access first happens and prompt the user to make a decision.

The third aspect of PMP's design is an easy to understand UI. When prompting the user to make a privacy protection decision, it should be done in a manner that is consistent with other notifications, for example mimicking the location access prompts in iOS 5. PMP prompts should therefore simply be an extension of any existing iOS privacy prompts. Next, since users may want to *temporarily* disable protection to use a specific app feature, for example location based search on an app, PMP should allow this without necessarily exiting the app. Finally, users should be able to view the privacy settings for all installed apps in one place, so they can browse and keep track of what decisions have been made.

The final design goal of PMP is the creation of a crowd-sourced recommendation engine to help users make more informed protection decisions, perhaps even automatically. The basic idea is to leverage the manual protection decisions made by users of a particular app, and use it to generate recommendations for other users of the same app as shown in Figure 1. The recommendation algorithm that PMP uses should account for developers trying to potentially game the system by providing initial recommendations, as well as give preference to users who make a lot of manual decisions while discounting users who make a handful of decisions. PMP should allow the recommendation algorithm, and hence the recommendations, to be changed on the fly as additional data and decisions are made. Using crowd-sourcing for recommendations on privacy settings was in part inspired by prior work which shows that less-experienced users can benefit from customizations decided by experts [18].

5. IMPLEMENTATION

In this section we describe our implementation of ProtectMyPrivacy on the iOS platform, highlighting how we support our design goals mentioned in Section 4. We first describe the features of iOS and the capabilities provided by jailbreaking that we utilize for PMP. Next we describe the PMP app itself that users install on their devices. Finally, we describe the back end infrastructure supporting PMP and our crowd sourced recommendation system.

5.1 iOS Platform Features

A background of iOS, its design and security features was provided in Section 2. iOS apps run as native binaries in an XNU environment which improves performance and also gives programmers access to familiar C++ APIs. One of the design goals of PMP was to support unmodified apps, which for iOS requires adding functionality to running processes. As mentioned in Section 2 it is not possible to do this due to iOS security features, without "jailbreaking" the device. On a jailbroken device, however, there is support for the Mac OS X DYLD_INSERT_LIBRARIES environment variable (similar to LD_PRELOAD on Linux) which allows plugins to be loaded before apps execute. This has enabled the creation of an open-source project MobileSubstrate (MSub) to standardize the iOS plugin API [12].

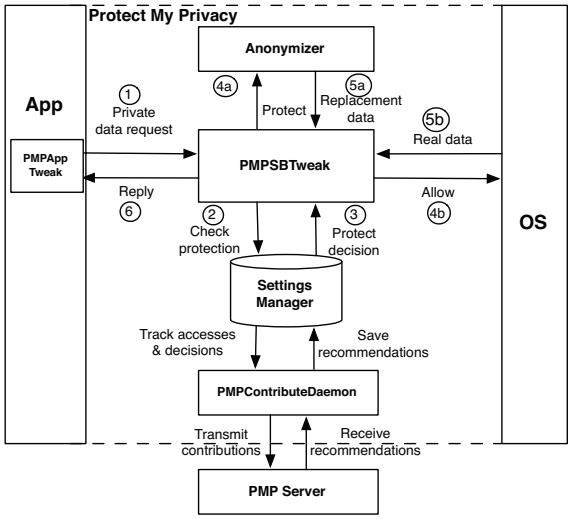


Figure 2: ProtectMyPrivacy acts as a layer between apps and the OS – the labels (1) through (6) denote the different steps taken when private information is accessed.

MSub allows developers to inject code into the address space of other iOS apps, replacing the existing implementation of specific classes. There are currently 100+ MSub powered customizations or 'tweaks' available for iOS in the Cydia Store. A popular one, for example, allows modification of the visual theme of applications such as changing color schemes and icons. MSub supports filters, which restrict what apps will be modified by a specified code injection. For example we specified that our dynamic libraries are only loaded into executables that link to the UIKit, the iOS UI framework; thus it is only loaded into executables that users actually interact with.

iOS APIs are a mixture of Obj C and C. The UDID and location APIs are in an Obj C framework, while contacts and music are in C; MSub provides method replacement APIs for both languages, however it still took a significant amount of effort on our part to determine which methods should be replaced to detect access to the various potentially privacy invasive APIs. For C, MSub uses a technique called *hooking* that works at the assembly level, and for Obj C, it uses *method swizzling* that uses the built-in `class_replaceMethod` function. PMP requires the plugins loaded into existing apps to communicate with a centralized PMP module for which we use the CPDistributedMessagingCenter, a wrapper on top of the Mach Interface Generator (MIG) generated Remote Procedure Call (RPC) subsystem. It uses a publish/subscribe API for transmitting hash-tables containing strings, with an synchronous mode to allow message senders to await replies.

5.2 PMP iOS App

We designed the PMP app which runs on an iOS device to be modular, with five components that communicate with each other via RPC. These are *PMPAppTweak*, *PMPSBTweak*, *PMPContributeDaemon*, *PMPWeeApp* and *PMPSettings*. Figure 2 illustrates the different PMP mod-

ules at a high level, as well as how they communicate with each other and the server.

Using MSub we implemented *PMPAppTweak*, a dynamic library that is loaded into all apps launched by the user. When this dynamic library is launched it replaces various methods used to access private information. In the case of the Address Book Framework (AB), for example, it uses `MSHookFunction` to replace `ABRecordCopyValue(...)` which is essentially a public API around a directly inaccessible SQLite database and returns the data for a particular property in an AB record. We monitor this method call for properties of interest, such as `kABLastNameProperty`, `kABPhoneProperty` or `kABEmailProperty`, and when detected, we activate our alternate version of the method. First, we check with the PMPSBTweak (see below) by means of RPC, on whether this app should have AB information protected. If so, we first invoke the original method call to retrieve the real AB information, then pass it to an anonymizer. Our anonymizer takes into account the dynamic nature of the AB where records do change, and rather than build a default fictitious address book, we randomly move around characters and the numbers in each record. Since we keep the same record count, our system is more robust to potentially serious array out of range errors as the application tries to access address book data. Furthermore, when we anonymize the data at runtime we use the same seed generated once per device which ensures the same modified value is passed to all apps, to give any cross-app tracking the illusion of consistent and valid data. We replace all other ABRecord methods in the same way so that the entire API is covered for providing replacement information. We note that it may be possible by a determined hacker to reverse engineer the original values by observing anonymized data from many devices. For those circumstances stronger anonymization schemes such as format-preserving encryption [2] may be better and we plan to explore them further. We are also considering the option of returning a set of preset names with random data for the address book. We currently protect the identifier, location and music in an identical way by replacing the appropriate method calls that provide access to these private data items. Our method is easily extensible to other private data types that may be accessed using iOS method calls and is not limited to the ones we currently protect. For example, we have recently added protection for the photo library but did not include it in this paper because we have not evaluated it yet.

We have implemented another MSub dynamic library, our *PMPSBTweak* module, that is loaded into the Springboard, the iOS home screen application that displays app icons and manages their launches, comparable to the desktop application in a PC environment. This module allows us to access the list of apps that are currently installed, look up their identifier and version numbers, and allow us to respond to notification when new apps are upgraded or installed. For example, on this notification we query our PMP server for updated recommendations. This module also acts as a central store for the PMP settings database. That is, it loads the settings on device boot, and ensures they are saved correctly. This module also listens for RPC messages from apps asking if private information of each type should be protected, and handles them as follows. First it checks to see if the user has previously decided to use recommendations for this particular app. If they have not and recommendations are available for the app, we prompt the user to check if they would like to use them (Figure 3c). This prompt enables the user to configure all the protection settings for this app at once. Alternatively, if no recommendation is available, then the user must respond to the privacy prompt by tapping "Protect" or "Allow" as shown in Figure 3a. The RPC message call from the source app causes it to pause while it waits for a response.

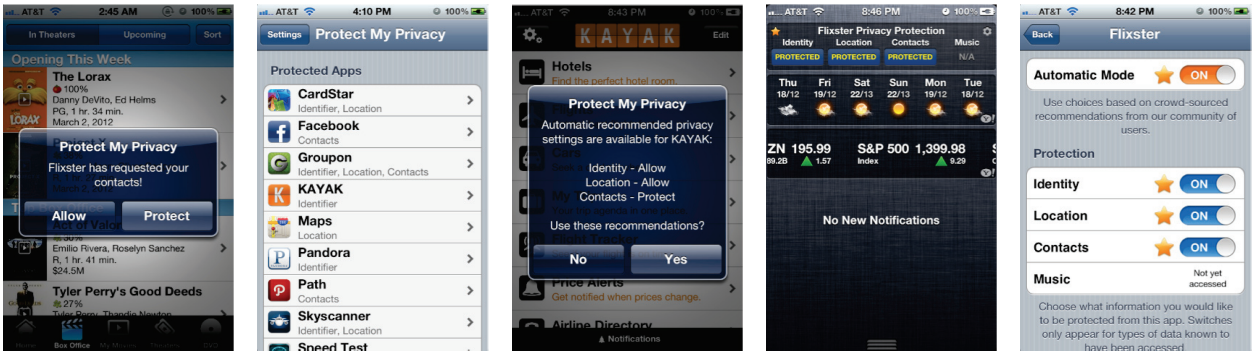
The third module is *PMPContributeDaemon*, a background daemon which persists across reboots. It can be started (or stopped) by the PMPSettings app if the user opts-in or out of contributing their PMP activity including protection decisions. When started this module listens for RPC messages from PMPSBTweak denoting changes to protection settings, as well as what private data each app is attempting to access – which are sent to our PMP server. Users that contribute their protection decisions are rewarded with recommendations for their other apps. Based on our modular design, all communication with the PMP server is confined to this daemon. As a result, if a user turns off contributions, this daemon is stopped and thus there is no server communication. If recommendations had been previously received, we leave them on the device although they do not get updated any further. Therefore, due to the way our system is designed, it is in the users interest to leave the contribute feature on in order to benefit from the recommendations feature.

PMPSettings, as shown in Figure 3b, is where PMP can be configured. When the PMP app is launched it displays the privacy settings of all apps configured so far. Users can review their previous protect or allow decisions, update them, or decide to start using the automatic recommendations as shown in Figure 3e. We also provide options to configure the replacement data, for example generating a new random unique identifier, and allowing the user to choose a replacement fake location on a map.

PMPWeeApp is a Notification Center (NC) WeeApp (or plug-in) that offers quick access to protection settings while an app is running. On iOS (v5.0 and later) users can drag their finger down from the top of the screen and they are presented with NC, which displays status information such as current weather, or new emails. NC supports plugins, although only unofficially at present, which enabled us to add an area to NC to manipulate the protection settings manually or switch to using recommendations without exiting the app (Figure 3d). *PMPWeeApp* is designed for users to temporarily allow access to their private data such as their location, while using a particular feature of an app, and then protecting it again afterwards. Of course, it is entirely possible for applications to cache private data as soon as access is granted and use it later.

Overall, we have implemented three places where users can configure PMP privacy settings: in the target app when the access is detected at runtime (InApp), in the Notification Center (NC) while the app is running, and in the PMP Settings app (Settings).

We have ensured that our implementation of the PMP app and its components do not impose any perceptible overheads in terms of performance by measuring the interactiveness and latency of different apps with and without PMP installed. Additional delay does occur when an app accesses a protected feature and the user is shown a popup to make a protection decision, however these are only to solicit user input on first access. It is also important to consider the



(a) (b) (c) (d) (e)

Figure 3: Screenshots of the ProtectMyPrivacy iOS application, from left to right: (a) in-app popup for a contacts access, (b) settings showing different applications and what features they have accessed, (c) Pop-up showing that a recommendation is available, (d) Notification Center showing which recommendations are in use (in yellow) for the Flixster app, and (e) Flixster settings in the settings app, showing stars for what is recommended, and also displaying if one of the privacy protected features has not been accessed yet.

impact of the PMP app on the battery lifetime of a device. While we have not yet performed detailed battery lifetime measurements, we do note that we have implemented PMP with energy efficiency in mind. For example, we measured the CPU usage of our *PMPContributeDaemon* which runs in the background to periodically upload data periodically to be negligible. Furthermore, to reduce the energy used by this daemon for network transfers we batch and compress data to reduce the frequency and the size of these uploads.

5.2.1 PMP App Security Analysis

By building PMP upon jailbroken devices and MSub, we lose compatibility with iOS code-signing enforcement which prevents runtime attacks, as covered earlier in Section 2. Hence when using PMP, users will have stronger protection against privacy-focused attacks, but lose protection against runtime attacks. Furthermore, it may be possible for an App Store app to detect jailbreak, download, then execute an unreviewed payload, which could potentially replace PMP's overridden methods back to their originals form. However, in that scenario an attacker could circumvent PMP entirely by reading files outside of the sandbox normally detected by Apple's runtime review. To limit these risks we added various integrity checks to PMP, and we recommend our users only install trusted apps from the App Store. The fact is, a jailbroken phone may be open to new security issues, a complete security analysis is beyond the scope of this paper. We merely use jailbroken devices as a method to experiment, develop, and collect data for our research.

5.3 PMP Server Software

We have installed a dedicated PMP server to collect users' PMP activity, run various back end analysis and finally to deliver recommendations. We first developed a generic logging framework, called Leo, that utilizes the Doctrine ORM, MySQL, PHP and Apache. Leo allows storage and retrieval of data in an extensible format using simple insert and query APIs, with transmissions optimized for minimizing mobile data use and secured using SSL. Leo uses 'layers' for app data types, for example, protection decisions, access detections, and recommendations, each stored in distinct layers that correspond to individual MySQL database tables. Access to layers is secured with individual read and write keys.

When the PMP app sends data to be stored in a layer, it is sent as an Apple Property List (PList), a serialized dictionary containing standard fields such as the title, subtitle and app version, along with created and updated timestamps. It also has a properties dictionary for custom fields - layers have the capability to draw out these custom property fields and insert them into database table fields, and optionally add indexes for faster queries. The Leo client library for iOS that we have implemented has a simple API for logging system data, and this is cached locally on the device and uploaded whenever an Internet connection is available, so no data is lost. Leo also features a default app stats layer that allows us to track app usage, device types and OS versions. In PMP, the Protect layer is for storing users protection decisions and has custom fields for *feature* (string for identifier, contacts, etc.), *protect* (boolean for user protected or allowed) and *recommended* (boolean for if this was a recommended decision). Note that all communication between iOS clients and our server is done over SSL encrypted HTTP messages.

5.3.1 Generating Recommendations

Once we receive the protection decisions from our contributing users, we use them to generate recommendations based on the following parameters. First, we only consider apps with a minimum number of users ($n > 5$). Second, we include decisions from only active users of an app (used for more than a week). Third, we include decisions only from users who have made decisions for a minimum number of other apps ($n > 10$ apps). We use these conditions to prevent developers from gaming the recommendation system and improve the quality of recommendations by factoring out naive users and users who stopped using the app. We then process the resulting decisions to calculate the percentage of users who protect/allow each feature, for example, identity or location, within each app. If 55%-100% of valid decisions are to protect, we will recommended protect, while if only 0%-45% of valid decisions protect, we will recommended allow. In this example, we call the 10% range between 45% and 55% a *deadband*, where a recommendation to protect or allow is too close to call. We chose a 10%

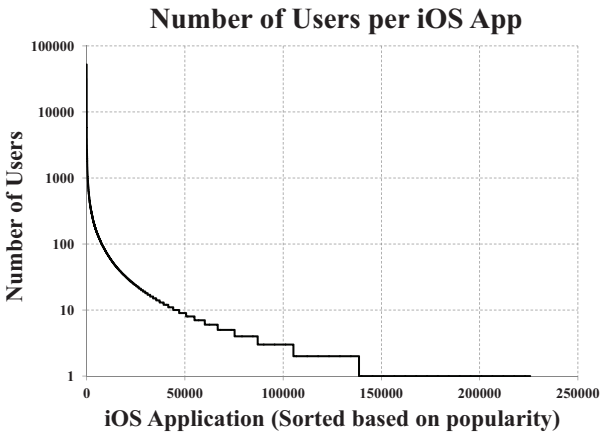


Figure 4: Number of Users per iOS app. Applications are sorted with the most popular app first.

Line	Features Accessed	Apps	Percentage
	Access **at least** 1		
1	– Identifier (I)	109,300	48.43%
2	– Location (L)	29,952	13.27%
3	– Contacts (C)	14,041	6.22%
4	– Music (M)	3655	1.62%
	Access **at least** 2		
5	– Identifier & Location	19,596	8.68%
6	– Identifier & Contacts	10,100	4.48%
7	– Identifier & Music	2,566	1.14%
8	– Location & Contacts	4,446	1.97%
9	– Location & Music	1,035	0.46%
10	– Contacts & Music	805	0.36%
11	Access All 4 (I,L,C,M)	94	.04%
12	Don't Access Anything	101,784	45.10%
13	Total Apps	225,685	100%

Table 1: Number of iOS apps that access certain protected features such as identifier, location, contacts or the music library.

deadband since it provided a good tradeoff between being able to provide recommendations for around 70% of the total apps, while reducing ones for which we were unsure about the right choice. For example, based on our current data, a 10% deadband provides recommendations for 69.6% of the total apps. Based on these calculations we created a recommendation layer in Leo (app & version) which our PMP app on iOS devices queries for recommendations.

For use in our evaluation later, we also generate an `expert` set of recommendations. While experts can be chosen in multiple ways, including perhaps even a set of paid 'certificate experts', we chose to designate users of our system who make a large number of protect or allow decisions, across a large number of apps used as experts. The intuition is that these users are active across many apps, and use apps often, to make better privacy decisions. We first identified the minimum number of decisions the top 2% of users of our system made (364) and also the the minimum number of applications the top 2% users of our system made decisions for (163). As a heuristic, we then define experts as ones who meet both criteria, i.e. they make a minimum of 364 decisions across at least 163 different apps, which results in 1133 users (1.2%) from our current user base of 90,621 users. We then use the same 10% deadband used for the general recommendation table, to create our `expert` recommendation layer in Leo based on these experts decisions. In Section 6 we evaluate the user acceptance rates of both the general and the expert recommendations.

6. EVALUATION

We released PMP via the Cydia Store for free in February 2012. Over the past nine months we have continued to iterate the design; releasing new versions with refinements and bug fixes as shown on our project website [21]. The first set of versions of our app (V1.x) did not deliver recommendations as we focused on scaling our system and collecting data about user choices on what they protected or allowed. In the third week of April we released an updated version with the crowd sourcing based recommendation feature enabled (V2.x). Subsequently, on Dec 3rd, 2012 we released another version (V 2.3.x) that included user surveys to help us analyze our users privacy choices and gather their feedback on recommendations.

In the rest of this section we present observations from our extensive dataset, as well as evaluate the effectiveness of PMP in protecting user privacy. We first describe the different user statistics we collect which form the basis of our quantitative analysis. We then present user and app statistics with a breakdown on what private pieces of data apps access, using this to evaluate the protect or allow decisions made by users. Lastly, we evaluate our recommendation engine and show its effectiveness.

6.1 Data Collected from Users

The PMP server receives data only from users who have contribute enabled in the PMP app. This data includes the app name, ID and version, what types of private information each app tries to access, and how many such requests happen. These statistics are collected on every app invocation and then batched and sent to our server. We also receive statistics from users about the decisions they make, either to protect or to allow, for each of the privacy sensitive information types. These protection statistics are collected for each app and also let us know where the decision was made, for example, in the in-app alert (InApp), in the PMP settings app (Settings) or using the iOS Notification Center (NC). These pieces of information are used by our recommendation engine feature.

6.2 User and Application Breakdown

Based on our analysis of the data we have collected at the time of writing this paper, we have 90,621 unique users. We determine the number of users based on the unique hashed UDIDs recorded in our database and also corroborated that with the number of downloads of our PMP software. Note that we account for users upgrading or resetting their device, and only count the number of unique hardware devices. Based on our dataset, we have observed 225,685 unique apps across all our users (considering multiple versions of an app as a single app). This number is less than a third of the estimated 775,000 total apps in the iOS App Store (as of January 2013), which suggests that two thirds of the App Store apps are not downloaded by our sample set of users, although we are confident that our dataset from 90,621 users

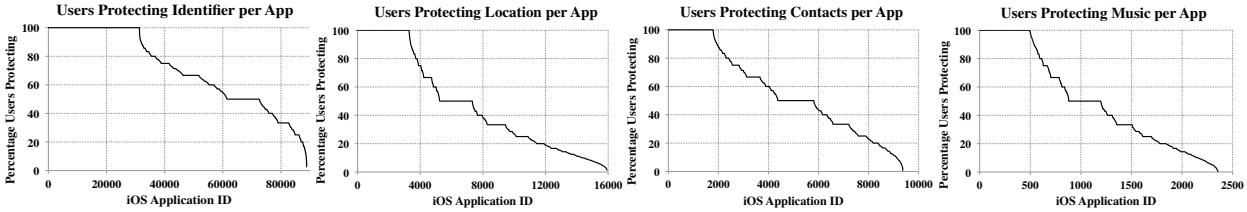


Figure 5: Percentage of users per app that protect access to (a) Identifier, (b) Location, (c) Contacts or (d) Music. The graphs are sorted by apps which have the highest percentage of users protecting access to the particular feature first.

statistically accounts for the most popular apps. Furthermore we noticed that within the 225,685 apps we observe, only 75,284 (33.3%) of them had at least five users, while the rest don't seem very popular.

In Figure 4 we plot the number of users (Y-axis in log scale) per unique app using PMP, showing the apps with the most users first. The distribution here follows a broadly log-linear distribution, correlating with other studies of smartphone application usage [4]. A few apps are very popular, with large number of users: 41 apps have over 10,000 users, 520 apps have between 1,000 - 10,000 users, while 7,134 apps have between 100 - 1,000 users. We see that there is a very long tail in the distribution: 39,365 apps have between 10 - 100 users, and finally 178,626 apps have less than 10 users each. Although our data is collected from jailbroken users, we nevertheless believe it is representative of regular non-jailbroken users for the reasons mentioned in Section 2. For instance, our 90,621 users install the same apps (particularly free apps) as users of non-jailbroken devices, which are only available in the regular iOS App Store. Note that while jailbreaking has become increasingly trivial, leading to an estimated 23 Million users jailbreaking their devices as of March 2013[24], it still requires an additional step and therefore our user population is likely biased towards more power users. Furthermore, our users who found and installed the PMP app from the Cydia jailbreak store are likely to be more privacy conscious than perhaps the average user.

6.2.1 Applications Accessing Protected Features

Next we analyze which apps accesses what privacy sensitive information types. Table 1 summarizes the breakdown for the number of apps that access at least one feature, ones that access at least two features, and ones that access all of the protected features. Our data shows some interesting trends. We see that 48.4% of the total apps access the identifier, 13.2% access location, 6.2% access contacts and only 1.6% access the music library (Line 1 to 4). When looking at the data for apps that access at least two protected features we observed that identifier and location (Line 5) are accessed together by 8.6% of apps while the identifier and the contacts (Line 6) are accessed by 4.5% of the total apps. This data points to the possibility that some of these apps may be legitimately accessing contacts, location and the identifier to provide a specific features such as "email friends" or to provide location-based services. As shown in prior work, a large number of apps often include third party ad frameworks which access privacy sensitive information to provide targeted ads [6, 14]. We present a specific app case study in Section 6.5. The focus of our work is not to identify which of these apps legitimately require access to the private data types. Instead, we rely on our users decision to protect or allow access, based on whether they think that an app should be granted access for its functionality, a user visible feature, or a clearly communicated need.

Statistic	Number
Users making 1 to 10 decisions	18,757 Users
Users making 10 to 100 decisions	44,260 Users
Users making 100 to 363 decisions	16,729 Users
Users making > 363 decisions (Top 2%)	1,865 Users
Users making decisions on 1 - 10 Apps	24,344 Users
Users making decisions on 10 - 50 Apps	36,907 Users
Users making decisions on 50 - 162 Apps	19,731 Users
Users making decisions on > 162 Apps (Top 2%)	1,854 Users
Average Number of decision across all our users	66 Decisions
Expert Users (> 363 Decisions & > 162 Apps)	1,133 Users

Table 2: Summary table showing the breakdown of the number of users making protect or allow decisions. A large number of our users make between 10 and 100 decisions.

6.3 Breakdown of Protection Decisions

As mentioned earlier, as PMP users make decisions to either protect or allow an app access to any privacy data, the PMP contribute daemon sends us that decision. Based on this collected data, on our server we can determine the distribution of the protect or allow decisions across different apps. Figure 5 (a) shows the percentage of users that decide to protect access to the identifier (I) across all the apps that access it. Note, the apps are sorted based on the percentage of users protecting access, thus the apps for which 100% of our users choose to protect are shown first. Figure 5 (b), (c) and (d) similarly show the percentage of users protecting access to location (L), contacts (C) and the music library (M) respectively across the apps that access these features. We make two key observations from these figures. First, we can see that for a significant fraction of apps, users unanimously choose to protect access to one or more of the four protected features (I,L,C,M). This behavior indicates that none of the users of these apps thought there was a reason for them to have access to their private data. Second, there are in general more apps for which users choose to protect, rather than allow access to three of the four features (I,C,M) except location where its the opposite with more apps being allowed access.

Table 2 provides a breakdown in terms of the number of users that make decisions, binned into different categories based on how often they make protect (or allow) decisions. These decisions are ones that users made themselves (be-

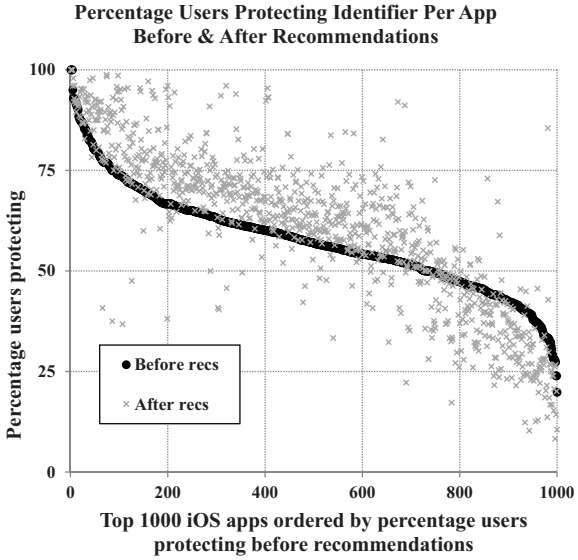


Figure 6: Percentage of users protecting (or allowing) access to their **identifier** before and after recommendations per app. Overall, after recommendations the percentage protected goes up for most apps since a large fraction of them have a protect recommendation for identifier.

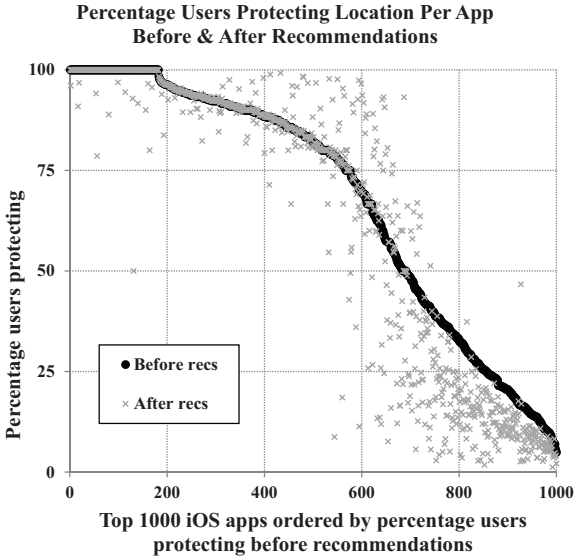


Figure 7: Percentage of users protecting (or allowing) access to their **location** before and after recommendations per app. As expected, the percentage protected goes up for apps with a protect recommendations and goes down for apps with an allow recommendation.

fore seeing recommendations) across all their apps and for accesses to either of the private data items, for example, identity or location. The users that make less than 10 decisions (18,757) are those who may have installed PMP and subsequently uninstall it after use, or users who turned contribute off so that we do not get their decisions. However, as can be observed from the table, an average user of PMP makes over 66 decisions, while a large number of our user population makes 10 to 100 (44,260 users) and 100 to 363 (16,729 users) decisions. Table 2 shows the benefit of allowing users to contribute their protection decisions, and their willingness to do so, since it is that drives our ability to provide recommendations.

6.4 Recommendation Feature

In the previous section we showed that many of our users manually make decisions to protect and allow individual apps access to privacy features using the PMP UI on their device. However, not all users necessarily want to make these individual decisions for each app they have installed. Furthermore, in some cases, users may not be convinced whether certain apps *should* have access to their private data, and whether that private data is essential for the app's functionality. We implemented our crowd sourced recommendations engine especially for these users and we next evaluate its effectiveness.

In order to observe the effect of our recommendations on the protection settings of apps, we considered the 1000 most popular apps which access the identifier including only those apps for which we have recommendations. Our intuition was that if our recommendations were having their desired effect, the number of users protecting their identifier for an app should increase if we recommend protecting, and correspondingly decrease if we recommend allowing. To validate this hypothesis, Figure 6 plots the percentage of users protecting access to the identifier before and after receiving recommendations for the top 1000 apps. For ease of understanding, we sorted the apps in decreasing order of the percentage of users that were protecting access before recommendations were available. As can be observed from Figure 6, for most of the apps the percentage of users protecting (or allowing) access to their identifier does indeed follow the recommendations. The percentage protected for apps with a recommend protect increases, while the percentage protected for apps with a recommend allow decreases after recommendations. There are few cases where this does not happen, for example, if the app has a sudden increase in the number of users who make manual decisions against the recommendations. We note that the next time the recommendations are generated those particular apps will likely see a change in their recommended settings. Figure 7 shows a similar plot for location before and after recommendations are available. We did not include plots for contacts or music in the interest of space.

Total Number of Apps (With at least 5 users)	52,386
Total Apps with Recommendations Available	36,442
Top 10K apps with Recommendations Available	9,709
Top 1K apps with Recommendations Available	966
Total Recommendations Shown to Users	1,935,213
Total Recommendations Accepted by Users	1,297,614
Percent Apps with Rec. Available	69.6%
Percent Top 10K Apps with Rec. Available	97.1%
Percent Top 1K Apps with Rec. Available	96.6%
Percent User Acceptance of Rec. Shown	67.1%

Table 3: Summarizing the effectiveness of our recommendation feature. Over 67% of almost two million recommendations offered to our users are accepted by them. We can also recommend protection settings for 97.1% of the 10,000 most popular apps.

	General	Expert	Matching Recs.
Apps with Recs. for Identity	85.0%	85.4%	74.2%
Apps with Recs. for Location	24.9%	25.2%	90.9%
Apps with Recs. for Contacts	9.7%	10.1%	86.1%
Apps with Recs. for Music	2.9%	2.9%	89.6%

Table 4: Comparing **General** and **Expert** recommendations. Overall, the recommendations are similar with 74.2% - 90.9% of the app recommendations matching for different features.

Next, we evaluate the overall efficacy of our recommendation feature by looking at the number of recommendations available and the overall acceptance of those recommendations by our users. Table 3 provides a summary of the recommendation feature based on data collected for V2 of our PMP app (all data before Nov 3rd, 2012). As can be observed, PMP can provide recommendations for 69.6% of the apps with at least 10 users, 97.1% of the top 10K Apps, and 96.6% of the top 1K most popular apps. As we gather more data for less popular apps (less than 10 users) we will be able to provide more recommendations. Furthermore, as can be seen from the table, the acceptance rate of all recommendations (almost two Million) shown to our users is 67.1% giving us confidence that the feature is indeed successful and liked by our users. Finally, we observe that over 99% of our users do have the recommendation feature turned on, denoting that they are indeed interested in receiving recommendations and find them useful.

6.4.1 General and Expert Recommendations

As described in Section 5.3.1 earlier, we generate an expert set of recommendations based on the data from the top 1% of our users in terms of the number of decisions and the number of apps they make decisions for. On December 3rd, 2012 we released V3 of our PMP App that uses these expert recommendations in addition to the general recommendations. Half of the users who have upgraded to this version of our app are shown general recommendations while the others are shown the expert recommendations. In addition, when users do not accept PMP recommendations we briefly asked them to select the reason for doing so. The goal of collecting this information was to identify *why* users reject recommendations coming from the crowd.

Table 4 shows a comparison of the general and expert recommendations. The first column shows the percentage of apps that have recommendations for protecting (or allowing) identity, location, contacts or music based on the general recommendations. Similarly, the second column shows the same values for the expert recommendations. The third column shows the actual overlap, or matching recommendations, for the applications for each feature. As can be observed from the table, a large fraction of the recommendations indeed match (74.2% to 89.6%) across the two sets of recommendations.

While our users do accept a majority of recommendations (67.1%), a natural question is why they don't accept the rest of the recommendations. Another related question is whether expert recommendations are accepted more than the general recommendations. As mentioned earlier, in the current version of PMP (V3), we show half of our users general recommendations while the other half are shown the expert recommendations. Furthermore, we asked the users to provide some context about why they reject a recommendation, also allowing them to click "Other Reason/Skip" if they did not want to give a reason. Table 5 provides a breakdown of the 10,000 or so recommendations that have been shown in the current version of the PMP app over over week of Dec 3rd - Dec 10th, 2012. As we can see, users receiving either set of recommendations are as likely to accept them (75%). Our users reject around 3% of the recommendations since they do not match what they want, showing that our recommendations are most likely correct. Finally, around 7% of our users marked that they prefer to make manual decisions, although since they have the recommendation feature turned on, they do view the recommendations. We believe that these users are in fact influenced by our recommendations but like to have manual control over their settings.

	General	Expert
Accept Recommendations	75%	73.8%
Reject Recommendations	25%	26.2%
Total	100%	100%
− > Reason: Recommendation incorrect	2.9%	3.2%
− > Reason: Prefer manual choice	7%	7.2%
− > Reason: Prefer to respond on access	1%	1.4%
− > Reason: Other / Don't say	14%	14.6%

Table 5: User responses for general and expert recommendations. Both recommendations are accepted around 75% of the time. 2.9% of the time users reject recommendations since they are incorrect, while 7% of the time it looks like they view the recommendations although still make manual decisions.

	Everyone	Non-Experts	Experts
Very Low - Low	11 (0.6%)	10 (0.6%)	1 (0.7%)
Low - Neutral	63 (3.7%)	60 (3.8%)	3 (2%)
Neutral	867 (50.4%)	798 (50.8%)	69 (46%)
Neutral to high	340 (19.8%)	313 (19.9%)	27 (18%)
High - Very High	440 (25.6%)	390 (24.8%)	50 (33.3%)
Total	1721 (100%)	1571 (100%)	150 (100%)

Table 6: User Responses to the question about iOS expertise on a scale of -10 to 10, 0 being neutral (or no response).

	Everyone	Non-Experts	Experts
Very Low - Low	5 (0.2%)	4 (0.2%)	1 (0.7%)
Low - Neutral	23 (1.1%)	23 (1.2%)	0 (0%)
Neutral	973 (46.3%)	911 (46.7%)	62 (41.3%)
Neutral - High	354 (16.9%)	325 (16.7%)	29 (19.3%)
High to Very High	745 (35.5%)	687 (35.2%)	58 (38.7%)
Total	2100 (100%)	1950 (100%)	150 (100%)

Table 7: User Responses to the question about their privacy consciousness on a scale of -10 to 10, 0 being neutral (or no response).

To perform a deeper analysis of the responses collected from our users, in the latest version of our app we also asked our users to rate themselves on a scale of -10 (Very Low) to 10 (Very High) on their iOS expertise and also their privacy consciousness. We allowed them to be neutral, or give no response, by entering zero on both the questions. We then split the responses into those that came from our top-1% experts and those that came from the set of non-experts to see whether there was an bias or a clear observable trends in how users rated themselves. Table 6 reports the responses for the iOS expertise question while Table 7 reports it for the privacy question. There are several interesting obser-

vations from these responses. First, our data shows that consistent with the popular belief, jailbreakers are indeed more privacy conscious and rate themselves high on level of iOS expertise. Second, many of the users who we do not rate in our 1% expert group still rate themselves high on both the survey questions. Therefore, relying on self-reporting for the choice of experts to drive our recommendation system is not optimal. We believe our mechanism of choosing these experts based on their contributed decisions work very well in practice.

6.5 Application Case Study: Flixster

When we released PMP, we noticed that several popular apps were being detected and reported for accessing private data. One of those is Flixster, a popular app that provides users reviews and recommendations for movies. Within a few weeks we had the developers of Flixster contact us that their users complained that PMP flagged their app as accessing their address book. The developers claimed that they did not access the address book, as verified by an internal code review. They claimed that their app was being incorrectly flagged by PMP and that we should immediately fix the problem. We performed a detailed analysis of Flixster version 5.2 on our end, using tools such as an intermediate web proxy and TCPDump to observe the data transmitted. Based on these forensics, we determined that it was a 3rd party ad library that iterated over all the contacts in the user's address book and sent back privacy sensitive information about the user such as the contacts, location and various pieces of demographic information such as education, age, gender, zip code and race. After we provided feedback to Flixster about our findings, we did not hear back from them, however, an updated version 5.3 of the app uses a different ad library that no longer invades privacy in this way.

7. DISCUSSION

In this paper we chose to generate recommendations based on either the privacy decisions of all our users or those made only by experts that we identified from our dataset. Another alternative to generating recommendations is to personalize them somewhat by segmenting users into different categories based on their privacy preferences. These preferences could be captured based on their past privacy decisions or based on survey questions. ProtectMyPrivacy could in that case provide a different set of recommendations for users that are more paranoid about their privacy as opposed to those that are less concerned. We plan to explore different strategies of generating personalized recommendations as part of our future work, however we note that its not yet clear whether they will have higher acceptance than our current strategy of considering all the decisions together.

A natural question about ProtectMyPrivacy is its utility to the general, non-jailbroken, users of iOS. Ideally, we would have liked to develop PMP in a way such that it could be released in the App Store. However, this is not possible without working closely with Apple themselves since some of the extensions we have made to the OS, e.g. hooking method calls that access private data, are not allowed under the App Store guidelines. Therefore, in September 2012 we developed a 'lite' version of PMP that provided non-jailbroken users an audit of their apps, and alert them if particular apps were known to access any private data items as well as display our recommendations for those apps. We designed PMP Lite to simply query our database and while it was not be able to protect users by substituting fake data, it could still help non-jailbroken users make decisions about which apps to use, and ones to avoid. Furthermore, users could even leverage our crowd sourced recommendations to make more informed privacy choices using the additional privacy controls introduced in iOS 6 (released Oct 2012) for contacts and location. We submitted PMP Lite to the App Store for review, and after several weeks of waiting we received a phone call from the Apple App review team that our PMP Lite application would not be accepted. After filing a formal request for the reason for rejection, we received another phone call from Apple which essentially said that no form of our app would be accepted since the review team had trouble with the "basic concept of your privacy app". This experience underscores the benefit of doing our research project on jailbroken users first.

We do believe that regular users of iOS can indeed benefit from knowing which apps are known to access privacy protected data, and whether other users of the app think those accesses are warranted. In the near term we are looking to implement a HTML5 based web application that can be accessed on any iOS device using a web browser. Users can then enter the name of any application that they are interested in and view its privacy summary based on crowd-sourced data from users of PMP.

8. CONCLUSION

In this paper we show that access to privacy sensitive information such as the unique identifier, or user location or even the address book is commonplace in iOS apps. We then present PMP, a privacy protecting architecture for iOS that not only notifies users of access to this sensitive information by individual apps, but also provides a mechanism to allow such accesses or deny them by substituting anonymized shadow data in its place. Next, PMP allows users to contribute their privacy decisions, based on which we have developed a novel recommendation feature to suggest protection settings to other users of the same apps. Our data shows that within a period of just nine months our community of users has grown to 90,621 people and they have contributed their protection decisions for 225,685 unique apps. Our analysis on our collected data shows several interesting findings. We observe that 48.4% of the total applications access the identifier, 13.2% access location, 6.2% access contacts and 1.6% access the music library. Next we find that a large number of our users actively make privacy decisions: 44,260 users make 10-100 decisions while 16,729 users make 100-363 decisions. We show that we can make recommendations for over 97.1% of the most popular (Top 10,000) apps, and over 69.6% of all apps with at least five users. Finally, we report that 67.1% of all recommendations given to users of our system were accepted by them. Our data also shows that as few as 1% of our users, classified as experts, make enough decisions to drive our crowd-sourced privacy recommendation engine. We believe that our data clearly points to the value and feasibility of using crowd sourcing as a method to help smartphone users make privacy decisions.

9. ACKNOWLEDGEMENTS

We would like to thank Barry Brown, and Sivasankar Radhakrishnan, Panagiotis Vekris and Thomas Weng, for their

comments on the early version of this paper. We would also like to thank our shepherd and our anonymous reviewers for their comments to help improve the paper. This work was supported in part by NSF grant SHF-1018632.

10. REFERENCES

[1] D. Barrera, H. Kayacik, P. van Oorschot, and A. Somayaji. A Methodology for Empirical Analysis of Permission-based Security Models and its Application to Android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, pages 73–84. ACM, 2010.

[2] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-preserving Encryption. In *Selected Areas in Cryptography*, pages 295–312. Springer, 2009.

[3] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. MockDroid: Trading Privacy for Application Functionality on Smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2011.

[4] M. Böhmer, B. Hecht, J. Schöning, A. Krüger, and G. Gernot Bauer. Falling Asleep with Angry Birds, Facebook and Kindle: A Large Scale Study on Mobile Application Usage. In *Proceedings of the International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI)*, 2011.

[5] A. Chaudhuri. Language-based Security on Android. In *Proceedings of the ACM SIGPLAN fourth workshop on Programming Languages and Analysis for Security (PLAS)*, pages 1–7. ACM, 2009.

[6] M. Egele, C. Kruegely, E. Kirdaz, and G. Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. In *Proceedings of the Network and Distributed System Security Symposium(NDSS)*, 2011.

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[8] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[9] W. Enck, M. Ongtang, and P. McDaniel. On Lightweight Mobile Phone Application Certification. In *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS)*, 2009.

[10] A. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011.

[11] A. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior. Technical report, University of California, Berkeley, 2012.

[12] J. Freeman. Mobile Substrate. `http://iphonedevwiki.net/index.php/MobileSubstrate`.

[13] A. P. Fuchs, A. Chaudhuri, and J. S. Foster. SCanDroid: Automated security certification of Android applications. *Manuscript, Univ. of Maryland, http://www.cs.umd.edu/~avik/projects/scandroidascaa*, 2009.

[14] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These Aren't the Droids you're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS)*, pages 639–652. ACM, 2011.

[15] Ian Shapira, The Washington Post. Once the Hobby of Tech Geeks, iPhone Jailbreaking now a Lucrative Industry, 2011.

[16] J. Jeon, K. Micinski, J. Vaughan, N. Reddy, Y. Zhu, J. Foster, and T. Millstein. Dr. Android and Mr. Hide: Fine-grained Security Policies on Unmodified Android. Technical report, University of Maryland, 2011.

[17] J. Lin, S. Amini, J. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and Purpose: Understanding Users Mental Models of Mobile App Privacy Through Crowdsourcing. In *Proceedings of the 14th ACM International Conference on Ubiquitous Computing (Ubicomp)*, 2012.

[18] W. Mackay. Patterns of Sharing Customizable Software. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 209–221. ACM, 1990.

[19] MobiStealth. `http://www.mobistealth.com/`.

[20] M. Nauman, S. Khan, and X. Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (CCS)*, pages 328–332. ACM, 2010.

[21] Protect My Privacy (PmP). iOS Privacy App. `http://www.protectmyprivacy.org`.

[22] N. Seriot. iPhone Privacy. In *Black Hat DC*, 2010.

[23] E. Smith. iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs). Technical report, Technical Report, 2010.

[24] The Next Web. Popular Jailbreak Software Cydia hits 14m Monthly Users on iOS 6, 23m on All Devices, March 2013.

[25] S. Thurm and Y. Kane. The Journal's Cellphone Testing Methodology. *The Wall Street Journal*, 2010.

[26] S. Thurm and Y. Kane. Your Apps Are Watching You. *The Wall Street Journal*, 2010.

[27] N. Y. Times. Mobile Apps Take Data Without Permission. http://bits.blogs.nytimes.com/2012/02/15/google-and-mobile-apps-take-data-books-without-permission/.

[28] M. T. Vennon. Android Malware: Spyware in the Android Market. Technical report, SMobile Systems, 2010.

[29] T. Vennon. Android Malware. A Study of Known and Potential Malware Threats. Technical report, SMobile Global Threat Center, 2010.

[30] XEUDOXUS. Privacy Blocker and Inspector. `http://privacytools.xeudoxus.com/`.

[31] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. Taming Information-Stealing Smartphone Applications (on Android). *Trust and Trustworthy Computing (TRUST)*, pages 93–107, 2011.